華東師範大學
EAST CHINA NORMAL UNIVERSITY

基于 SVM 的数据识别

姓名	学号	Presentation 题目	贡献
陈彪	51174500004	Support Vector Machine	程序实现，调参，论文撰写
王佩芬	51174500048	Hidden Markov Model	调参，论文撰写
赵丽梅	51174500061	Support Vector Machine	数据收集，论文撰写

摘要

本文基于支持向量机 SVM 模型分别对鸢尾花数据集 IRIS 和 MNIST 手写字体数据集进行了识别。本文基于 python 语言，使用机器学习包 sklearn 进行了实验。通过对参数 gamma 和 C 的调整，给出了对应的识别准确率数据。本文选择 kernel＝'poly'时，调整 degree 来查看分类的准确率。另外，通过查阅文献，在核函数的选择上，高斯核函数在分类准确率上有较好的效果，本文也通过实验验证了这个结论，在同样的条件下，高斯核函数 rbf 的识别准确率高于线性核函数 linear。本文也使用了其他的核函数进行了对比实验，对比了其他的模型在数据分类上的准确率。最后，提出了改进措施。

引言

支持向量机 SVM 模型在分类上有广泛的应用。鸢尾花数据集 IRIS 和手写字体数据集 MNIST 是测试分类算法的准确率常用的两个数据集，并且是小数据集和大数据集的代表。本文基于 SVM 模型对上述的两个数据集进行分类准确率的实验，实验基于 python 的 sklearn 包中的 svm 进行。本文根据具体的实验，将从以下几个方面介绍实验的工作。第一部分：问题描述，本部分将解释使用 SVM 进行数据分类的原因。第二部分：数据来源和处理，该部分将介绍 IRIS 数据源和 MNIST 数据源的信息，同时会着重介绍后者的处理方式。第三部分：在实验中使用的包，由于本文使用 python 语言进行建模，这部分会介绍数据处理包 numpy 和 pandas，可视化包 matplotlib， 机器学习包 sklearn。第四部分：建模过程和分析，该部分会介绍如何根据处理好的 IRIS 数据和 MNIST 手写字体数据使用 sklearn 中的 SVM 来进行建模，同时介绍了建模的调参过程。同时，介绍了不同参数下的耗时的信息，并做了简要的分析。该部分还展示模型运行结果显示的屏幕截图。第五部分：改进措施，该部分针对本实验提出了一些改进措施。

1 问题描述

问题背景：

手写体数字识别是图像处理与模式识别中具有较高实用价值的研究热点。手

写体数字识别就是让计算机模拟人自动识别纸张上的手写体阿拉伯数字。传统的手写体数字识别技术如人工分类、神经网络、决策树等识别方法普遍存在识别速度较低、识别准确率不高等问题，因此本实验使用了一种基于支持向量机(Support Vector Machine，SVM)的快速手写体数字识别方法。

该方法通过各类别在特征空间中的可分性强度确定 SVM 最优核参数，快速训练出 SVM 分类器对手写体数字进行分类识别。由于可分性强度的计算是一个简单的迭代过程，所需时间远小于传统参数优化方法中训练相应 SVM 分类器所需时间，故参数确定时间被大大缩减，训练速度得到相应提高，从而加快了手写体数字的识别过程，同时保证了较好的分类准确率。通过对 MNIST 手写体数字库的实验验证，结果表明该算法是可行有效的。IRIS 数据集是一个测试分类模型准确性的小数据代表数据集。使用 SVM 模型来对 IRIS 进行分类也达到了较好的效果。

2 数据来源和处理

本部分主要介绍了数据来源或者搜集过程，数据量，数据的预处理手段等。

2.1 MNIST 数据集

MNIST（Mixed National Institute of Standards and Technology database）数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST)。训练集 (training set) 由来自 250 个不同人手写的数字构成，其中 50% 是高中学生，50% 来自人口普查局 (the Census Bureau) 的工作人员，测试集(test set) 也是同样比例的手写数字数据。

MNIST 数据集可在 http://yann.lecun.com/exdb/MNIST/获取，它包含了四个部分：

Training set images: train-images-idx3-ubyte.gz (9.9 MB，解压后 47 MB，包含 60,000 个样本)；

Training set labels: train-labels-idx1-ubyte.gz (29 KB，解压后 60 KB，包含 60,000 个标签)；

Test set images: t10k-images-idx3-ubyte.gz (1.6 MB，解压后 7.8 MB，包含 10,000 个样本)；

Test set labels: t10k-labels-idx1-ubyte.gz (5KB，解压后 10 KB，包含 10,000 个标

签)。

MNIST 数据集包含 70000 张手写数字的灰度图片，其中每一张图片包含 28 * 28 个像素点。数据集被分成两部分：60000 行的训练数据集（MNIST.train）和 10000 行的测试数据集(MNIST.test)。其中，60000 行的训练集分拆为 55000 行的训练集和 5000 行的验证集。60000 行的训练数据集是一个形状为 [60000,784] 的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于 0 和 1 之间。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于 0 和 1 之间。

每一张图片都有对应的标签，标签是介于 0 到 9 的数字，也就是图片对应的数字，称为 "one-hot vectors"。一个 one-hot 向量除了某一位的数字是 1 以外其余各维度数字都是 0。例如，数字 3 将表示成一个只有在第 3 维度（从 0 开始）数字为 1 的 10 维向量。比如，标签 0 将表示成 ([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])。因此，其标签是一个 [60000, 10] 的数字矩阵。

2.2 MNIST 数据集处理

MNIST 手写体数字库是大小为 28px*28px 的图片数据，所以需要将手写体数字图像转换成向量。用 numpy 向量来表示图像数据，以便训练和测试模型。首先，将图像数据灰度化处理，每个像素点用一个灰度值表示。在这里，我们就将 28*28 的像素展开成一个一维的行向量，这些行向量就是图片数组里的行（每行 784 个值）。然后将其存入 numpy 数组中。依次解析所有的图片数据，得到了 60000*784 的一个二维矩阵。

2.3 IRIS 数据集

IRIS 数据集是常用的分类实验数据集，首次出现在著名的英国统计学家和生物学家 Ronald Fisher 1936 年的论文《The use of multiple measurements in taxonomic problems》中，由 Fisher 收集整理。IRIS 也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

数据链接参见：http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

3 使用的包

本实验主要使用了 pandas、numpy、matplotilb、sklearn 包。

3.1 Numpy

Numpy 是 Python 的一个扩展包，语法和 Matlab 有很多相似之处。它支持高维数组和矩阵运算，也提供了许多数组和矩阵运算的函数。另外，它在数组和矩阵运算方面速度很快，效率很高，对数组的运算都可以算在每个元素上。Numpy 提供了许多高级的数值编程工具，如矩阵数据类型、矢量处理，以及精密的运算库，专为进行严格的数字处理而产生。

3.2 pandas

pandas 是基于 Numpy 构建的含有更高级数据结构和工具的数据分析包，类似于 Numpy 的核心是 ndarray，pandas 也是围绕着 Series 和 DataFrame 两个核心数据结构展开的。Series 和 DataFrame 分别对应于一维的序列和二维的表结构。Series 可以看做一个定长的有序字典，基本任意的一维数据都可以用来构造 Series 对象。DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型，基本上可以把 DataFrame 看成是共享同一个 index 的 Series 的集合。

3.3 matplotilb

matplotilb 是 python 的绘图包，与 matlab 的绘图功能类似。matplotlib 是第一个 python 可视化程序库，有许多别的程序库都是建立在它的基础上或者直接调用它。使用 matplotlib 可以绘制条形图，散点图，直方图，箱线图等等。本实验在调试参数的时候，使用该软件包进行了绘图。

3.4 sklearn

sklearn 是基于 Numpy，SciPy 和 matplotlib 的一个机器学习算法库，设计的非常优雅，它让我们能够使用同样的接口来实现所有不同的算法调用，里面对一些常用的机器学习方法进行了封装，在进行机器学习任务时，并不需要每个人都实现所有的算法，只需要简单的调用 sklearn 里的模块就可以实现大多数机器学习任务。本文使用的 SVM 是机器学习算法库里面的一个重要的函数。

4 建模过程分析

4.1 数据的特征提取及分析

本文处理 IRIS 数据集的时候，将数据集的每条记录处理成一个数组，花的分类处理成一个标签。这两者是对应的，这个数据就是需要的关键参数。本文在选择训练数据和测试数据的时候，对每组随机选取了 45 条作为训练数据，剩下的 5 条作为测试数据。对于 MNIST 数据集，本文将图片对应的数据解析成对应个数的(28*28,1)的向量，再灰度化处理。对应的数字标签就是一个数，例如 1，2，3 等。这样，对于训练数据，我们会得到一个 60000*784 的一个数组，对应的是 60000 个 0-9 数字。这两个数据，即图片对应的 28*28 的灰度数据以及对应的标签就是关键的特征，测试数据对应的这两部分也是关键的特征。在计算准确率的时候，使用训练数据的图片对应的灰度化数据使用机器进行训练得到训练的结果，与训练数据的标签进行比较得到准确率。

4.2 调参

查看一个本文在进行 IRIS 数据集的训练过程中返回的参数信息：

SVC(C=1.0,cache_size=200,class_weight=None,coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',max_iter=-1,
probability=False, random_state=None, shrinking=True,tol=0.001, verbose=False)

从上面的返回信息可以看到，在默认状态下的参数以及参数的具体数据。下面是这些参数中的一些参数的介绍。

C：C-SVC 的惩罚参数 C，默认值是 1.0

C 越大，相当于惩罚松弛变量，希望松弛变量接近 0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C 值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

kernel ：核函数，默认是 rbf，可以是‘linear’， ‘poly’， ‘rbf’， ‘sigmoid’，‘precomputed’

0 – 线性：u'v

1 – 多项式：(gamma*u'*v + coef0)^degree

2 – RBF 函数：exp(-gamma|u-v|^2)

3 –sigmoid：tanh(gamma*u'*v + coef0)

本实验中默认使用的是 rbf 函数，本文也尝试使用 linear 函数，后文实验中给出了将核函数修改成 linear 的时候的准确率。根据 MNIST 数据集页面上指示的结果数据，本文同样测试了 kernal 为 poly 的时候的准确率。

degree ：多项式 poly 函数的维度，默认是 3，选择其他核函数时会被忽略。本文在实验中选用了核函数为多项式函数。并赋予其不同的参数。

gamma ：'rbf'，'poly' 和'sigmoid'的核函数参数。默认是'auto'，则会选择 1/n_features

coef0 ：核函数的常数项。对于'poly'和 'sigmoid'有用。

probability ：是否采用概率估计。默认为 False

shrinking ：是否采用 shrinking heuristic 方法，默认为 true

tol ：停止训练的误差值大小，默认为 1e-3

cache_size ：核函数 cache 缓存大小，默认为 200

class_weight ：类别的权重，字典形式传递。设置第几类的参数 C 为 weight*C(C-SVC 中的 C)

verbose ：允许冗余输出。

max_iter ：最大迭代次数。-1 为无限制。

decision_function_shape ：'ovo'， 'ovr' or None， default=None3

random_state ：数据洗牌时的种子值，int 值

主要调节的参数有：C、kernel、degree、gamma。本文针对上面的一些特点，主要对参数 C，kernel，degree，gamma 进行了调整，而且是采用了控制变量的方法，在其余参数不变的条件下，单独改变一个参数，查看在该参数变化的时候准确率的变化情况。

4.3 模型的调整

本文基于原始的 SVM 模型进行训练，在模型训练的过程中，多次调整其中的参数，主要是对其中的 C，kernel，degree，gamma 四个参数进行调整，在模型本身上并未进行大的调整。

4.4 结果分析

4.4.1 IRIS 数据集

本文使用 sklearn 中 SVM 默认设置的参数，分别使用函数 loadIRIS()和 loadflags()加载数据和标签。使用训练数据 trainIRIS.txt 分别获取到对应的数据和标签，然后用 SVM 的机器进行训练。

```
 def loadIRIS(p):
    f=open(p,'r')
    lines=f.readlines()
    datamat=[]
    for line in lines:
        a=[]
        for i in range(len(line.split(','))-1):
            a.append(float(line.split(',')[i]))
        datamat.append(a)
    return np.array(datamat)
'''获取数据集的标签信息'''
def loadflags(p):
    f=open(p,'r')
    lines=f.readlines()
    flags=[]
    for line in lines:
        flags.append(line.strip().split(',')[-1])
    return flags
```

查看训练得到的机器：

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

从上面的参数，结合前面的调参的分析，可知道，这里 C=1,0，核函数为'rbf'，

gamma='auto'，然后我们使用这个机器来训练数据，查看准确率：

```
In [13]: print("训练的精确度是: ",result)
训练的精确度是:  1.0
```

这里着重介绍的是，训练数据是原始数据中随机抽出 45 项，测试数据为剩下的 5 项组成的数据集。从这里看出，使用 SVM 算法，训练的准确率达到了 100%。

4.4.2 MNIST 数据集

针对 MNIST 数据集，我们首先使用两个函数将数据处理成对应的数字。分别是图像对应的数据和图像对应的标签。下面两个函数分别是处理的函数：

```
def get_images(filename, bol=False, length=10000):
    # Parameters -
    #    1. filename – FORMAT: filepath/filename
    #    2. bol - (default -False)-- get images for full length or not
    #    3. length of input images (default=10000)
    length = length*784
    with open(filename,'rb') as f:
        byte_=f.read()
        i = 16
        data = []
        while True:
            byte = byte_[i:i+1]
            if len(byte) == 0:
                break
            if i == length+16 and bol==False:
                break
            val = int.from_bytes(byte,byteorder='big', signed=False)
            data.append(val/255)
            i=i+1
    return data
```

```
# Input Lables 这是 MNIST 数据集的标签处理函数
def get_labels(filename):
    # Parameters -
    #    1. filename – FORMAT: filepath/filename
    with open(filename,'rb') as f:
        byte_=f.read()
        i = 8
        data = []
        while True:
            byte = byte_[i:i+1]
            if len(byte) == 0:
                break
            hexadecimal = binascii.hexlify(byte)
            decimal = int(hexadecimal, 16)
            data.append(decimal)
            i = i+1
    return data
```

在这两个函数的基础上，将 MNIST 数据集中的训练数据和标签解析后进行训练，得到相应的机器。在参数未经任何调整的情况下。得到的机器参数为：

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

训练得到的准确率为： 0.9446

```
In [26]: print("训练的精确度是: ",result)
训练的精确度是:  0.9446
```

查看得到在 sklearn 中的 SVM 的调参的方式，我们首先将核函数修改为’linear’.clf = svm.SVC(kernel='linear')，训练得到的准确率为：94.04%,可见比原

始情况下准确率低了。

本文继续选择调整核函数 kernel,这次将核函数选择为'poly',degree=2，即 clf = svm.SVC(kernel='poly',degree =2,gamma=0.03)，准确率为 98.07%。

```
模型训练中......
模型训练完成......
Succeed!
测试进行中......
测试完成......
训练的精确度是:  0.9807
339.07015051967005
```

核函数选择为 'poly',degree=3, 即 clf = svm.SVC(kernel='poly',degree =3,gamma=0.03)，准确率为 97.83%。

```
模型训练中......
模型训练完成......
Succeed!
测试进行中......
测试完成......
训练的精确度是:  0.9783
377.6526974225342
```

选择的 degree 为 4，gamma 为 0.03，即 clf = svm.SVC(kernel='poly',degree =4,gamma=0.03)，得到的准确率为 97.43%。在此基础上，查看 MNISTS 中的 SVM 方法的介绍，在核函数为'poly' 的时候，准确率应该能达到较高的水平，我们修改 degree，设置 degree=5，也即 clf = svm.SVC(kernel='poly',degree =5,gamma=0.03)，测试准确率达到 96.67%；再次修改 degree=9，也即 clf = svm.SVC(kernel='poly',degree =9,gamma=0.03)，这时得到的准确率为 92.49%，可见在核函数为'poly'的情况下，gamma=0.03，选择 degree 为 2 的时候准确率达到了较大的 97.43%。下图是在 kernel 为'poly'，gamma=0.03，degree 为 1 到 9 变化的时候对应的准确率的变化曲线。

n	accuracy	time/s
1	0.9368	727.4008606
2	0.9806	442.5626865
3	0.9787	455.9989356
4	0.9734	482.4702491
5	0.9658	526.9926231
6	0.9581	590.4072423
7	0.9469	661.8732531
8	0.9358	744.2327274
9	0.9254	835.5411348

表 4-1　kernel='poly',degree 从 1 变化到 9 对应的准确率和耗时(s)

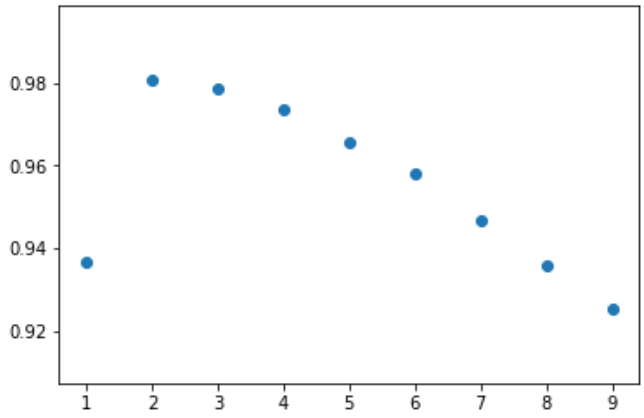

图 4-1　kernel='poly',degree 从 1 变化到 9 对应的准确率曲线

从上面的 degree 从 1 变化到 9 的时候，可见当 degree 为从 1 到 2，准确率变大，达到 98.06%；当 degree 从 2 变化到 9，测试的准确率变小。通过分析可知，应该是 2 多项式比较好，当多项式的次数变大导致过拟合导致准确率下降。

本文尝试使用核函数为 sigmoid 的时候，测试的准确率很低，仅仅 44.89%，故本文并没继续使用该核函数调整其他参数。

本文通过调研发现，参数 kernal 选择'rbf'较为明智的，根据 Andrew Ng 的相关机器学习的讲稿文献，其在训练模型时，选择的核函数均为 'rbf'，本文基于这一点，重点调整两个参数 gamma 和 C。下面的实验分别是在核函数为'rbf'，固定 gamma=0.03 改变 C 和固定 C=100 改变 gamma 和的情况下对应的训练时间(s)和准确率。

首先是在 gamma=0.03 的时候改变 C，C 从 0.01 变化到 20，准确率是增加的。在 C=20 之后，准确率不再改变，达到了 98.57%的较高值，实验结果如表 4-1 和图 4-1 所示。参考 MNIST 数据集中的介绍，在高斯核，SVM 算法对 MNIST 数据的分类误差率为 1.4%，可见准确率达到了传统 SVM 算法的实验最高值。从训练时间上，在 C 比较小的时候，耗时很长，准确率也不好，可见 C 较小是一种较不经济的选择。在 C 从 1 开始后，训练与测试的耗时变化不大，可见模型对 C 在 1 之后不敏感。这对实验控制变量选择 C 固定(例如 C=100)提供了依据。

c	accurac	time/s
0.01	0.9272	3888.496444
1	0.9845	901.3955707
20	0.9857	917.9576934
40	0.9857	907.6828448
60	0.9857	914.2598001
80	0.9857	928.6571328
100	0.9857	902.9927095

表 4-2　gamma=0.03，C 改变对应的准确度和耗时

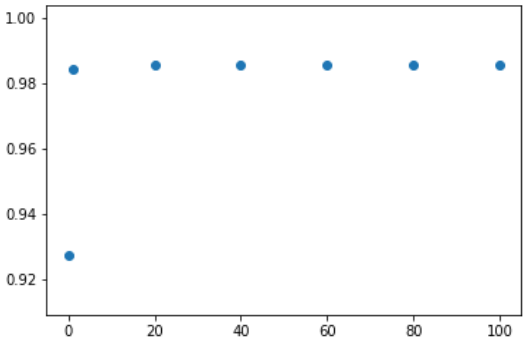

图 4-2　gamma=0.03，C 与准确度图示

接下来我们对 C=100，gamma 改变的情况下的准确度和耗时进行实验，并得到如下的结果，如表 4-2 和图 4-2 所示。

g	accurac	time/s
0.01	0.9824	540.8598496
0.02	0.9854	702.4880912
0.03	0.9857	1036.33215
0.04	0.9851	1296.602721
0.05	0.9837	1696.893731
0.06	0.982	2395.129667
0.07	0.9788	3926.446304
0.08	0.9746	6433.979788
0.09	0.9673	8893.506456
0.1	0.957	11105.11564

表 4-3　C=100，gamma 改变对应的准确度和耗时

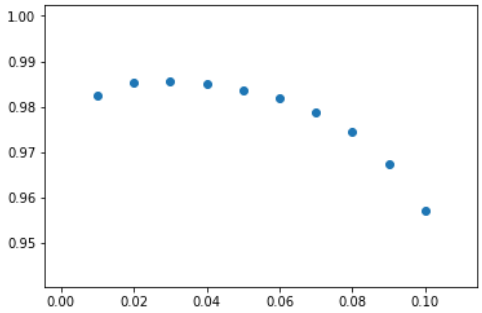

图 4-3　C=100，gamma 改变对应的准确度

从上面的实验结果可以看到，在 C=100 时，随着 gamma 从 0.01 改变到 0.03，模型的准确率在增加，并在 gamma=0.03 的时候达到最大 98.57%，随后，随着 gamma 的增大，准确率下降，同时训练耗时大大增加。这表明随着 gamma 的增大，计算的消耗大大增加，并且训练效果下降。通过上面两个实验，可见 gamma=0.03，C 取 1-100 之间，模型的准确率达到最大 98.57%。

本实验的数据以及相关的代码文献在文末 github 的地址上都有存储。

4.5 和其他模型的对比分析

本文同样使用了其他的方法来进行对比，选择常用的 KNN 模型来处理 MNIST 数据集。使用 KNN 模型，使用相同的数据。本文调整了 KNN 中的参数 K，分别置 K 为 5，6，10，得到的分类准确率为 96.88%，96.77%，96.65%，实验结果如下图所示。可见使用 KNN 模型对手写字体进行识别时，不同的 K 会影响分类的准确率，另外使用 KNN 模型做分类的准确率未能达到 SVM 模型下的最佳准确率。使用 KNN 模型处理 IRIS 数据集的准确率达到了 100%。

本文尝试使用决策树模型，对数据集进行分类，在 IRIS 数据集上，处理的准确率达到了 100%，但在处理 MNIST 数据集的时候，准确率仅仅达到了 87.89%，同时还比较耗时，实验结果如下图所示。可见，如本文前述，决策树算法在进行大数据集数据分类的时候识别准确率确实不高。

```
In [46]: from sklearn.neighbors import KNeighborsClassifier
    ...: neigh = KNeighborsClassifier(n_neighbors=5)
    ...: neigh.fit(train_data, train_labels)
    ...: neigh.score(test_data,test_labels)
Out[46]: 0.96879999999999999
```

图 4-4　KNN 算法分类 MNIST 数据选择 K=5 的训练准确率

```
In [9]: from sklearn.neighbors import KNeighborsClassifier
   ...: neigh = KNeighborsClassifier(n_neighbors=6)
   ...: neigh.fit(train_data, train_labels)
   ...: neigh.score(test_data,test_labels)
Out[9]: 0.9677
```

图 4-5　KNN 算法分类 MNIST 数据选择 K=6 的训练准确率

```
In [8]: from sklearn.neighbors import KNeighborsClassifier
   ...: neigh = KNeighborsClassifier(n_neighbors=10)
   ...: neigh.fit(train_data, train_labels)
   ...: neigh.score(test_data,test_labels)
Out[8]: 0.96650000000000003
```

图 4-6　KNN 算法分类 MNIST 数据选择 K=10 的训练准确率

```
In [10]: from sklearn import tree
    ...: clf = tree.DecisionTreeClassifier()
    ...: clf.fit(train_data, train_labels)
    ...: clf.score(test_data, test_labels)
Out[10]: 0.87890000000000001
```

图 4-7　决策树算法分类 MNIST 数据的训练准确率

5 改进措施

本实验使用了 SVM 模型来对 IRIS 和 MNIST 数据集进行分类，并通过分类准确率来判断分类的好坏。本文根据具体的调研同时根据实验的结果查看，发现不同参数对应着不同的分类准确率。所以本模型可以考虑多一些组合,或者是使用一个启发式方案自动找到最佳的参数组合。通过参考相关的文献，我们了解到，使用 SVM 进行 MNIST 数据的分类的最优错误率达到了 0.56%，本文的实验与 MNIST 数据集最初的错误率 1.4%尚有 0.03%的差距，所以本文后续改进应该是参阅相关文献来修改 SVM 的参数。

本实验在数据选择上，使用 IRIS 数据集，可能存在着训练数据较大，测试数据规模较小的问题，导致使用 SVM 模型尚未调参的时候准确率达到了 100%，后期可做的是修改测试数据与训练数据的规模，或者是使用交叉验证(CV)的方法来修改测试数据与训练数据从而使得能更好地反应模型在分类准确率上的好坏。针对 MNIST 数据集，该数据集规模庞大，实验中也发现这导致计算耗时很长，所以，改进的方案是首先使用主成分分析(PCA)算法来将数据降维，保留反应数据特征的大部分再使用 SVM 进行分类，从而达到在保持分类准确率的情况下，

减少资源时间消耗。

总结

本文基于支持向量机 SVM 模型分别对鸢尾花数据集 IRIS 和 MNIST 手写字体数据集进行了分类并给出了分类准确率。在未调参的情况下，前者识别准确率达到了 100%，后者识别准确率得到了 94.46%。经过对参数 gamma 和 C 的调整，在 gamma=0.03，C=100 的情况下手写字体识别准确率达到了较高的 98.57%。同时本文对不同的 gamma 和 c 进行了实验，给出了对应的识别准确率数据。本文也调整了参数 kernel，分别使用了'poly'，'linear'，'rbf'等核函数，并针对不同核函数的特点对应调整了先关的参数，在核函数为'poly'时,调整 degree，给出了相应的准确率的图像，发现核为'poly'，degree 为 2 时候，准确率达到 98.07%，同时分析这种情况产生的原因是多项式次数较低会导致欠拟合，多项式次数较高会导致过拟合；核函数为'rbf'时，选择 gamma＝0.03 可以使得准确率达到 98.57%。本文通过 KNN 模型和决策树模型进行对比，发现使用 SVM 模型分类准确率好于前两者。

参考文献

1. 李航《统计学习方法》
2. 周志华《机器学习》
3. Peter Harrington《Machine Learning in Action》
4. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
5. http://yann.lecun.com/exdb/MNIST/
6. http://scikit-learn.org/stable/index.html
7. https://github.com/marinajacks/Machine-Learning/tree/master/PCA

代码信息：https://github.com/marinajacks/SVM-MNISTS.git